# [1.@Override](mailto:1.@Override)

准确的覆写:如果没有覆写或者实现(拼写错误等)成功,编译器会察觉不到,但是加上了Override后,编译器就能检测到,如果没有覆写或实现成功,就报错

|  |
| --- |
| class Book{  @Override  public String toString() {  return "haha";  }  } |

# 2.@Deprecated

声明过期操作

|  |
| --- |
| public class test{  public static void main(String args[]){  Book b = new Book();  b.~~fun~~(); //表示这个方法过期了  }  }  class Book{  @Deprecated  public void fun(){}  } |

# 3.@SuppressWarnings

压制警告,可以压制多个警告

|  |
| --- |
| public class test{  @SuppressWarnings({"unchecked"}) //放在函数外面  public static void main(String args[]){  Book b = new Book();  b.setTitle("helllo"); //如果不压制警告,这里会有黄色的颜色  }  }  class Book<T>{  private T title;  public void setTitle(T title) {  this.title = title;  }  } |

# 4.注解的作用

Annotation

注解:给程序带上一些标记,从而影响编译器运行程序的结果

注解的作用:

1. 可以在程序上(类,方法,属性)携带信息
2. 注解简化(取代)配置文件(xml或properties)

每个框架都有两种方案:

1. 把参数放在xml方法
2. 把参数放在注解中

# 5.自定义注解

注解细节:

1)属性的类型可以为基本数据类型,也可以是数组类型

2)使用default关键字给注释一个默认值

3)当注解中使用value名称的属性,则可以省略“value=”不写

4)一个方法可以带多个注解

Author.java:

|  |
| --- |
| public @interface Author {  //声明属性  String name() default "lw";  String modifiTime() default "2017-1-19"; //default后面接默认值  String[] address();//带有数组类型的属性  //如果注解的属性名称为value,并且没有其他的属性,可以不用写value=xxx,直接写xxx就可以了  String[] value();  } |

# 6.元注解

[1.@Target](mailto:1.@Target):声明注解的使用范围

@Target({ElementType.METHOD,ElementType.FIELD})

METHOD:用方法上

FIELD:注解可以用在属性上

TYPE:注解可以用在类上

PARAMETER:用在参数声明上面

CONSTRUCTOR:用在构造函数上面

LOCAL\_VARIABLE:用在本地变量上面

[2.@Retention(RetentionPolicy.SOURCE)](mailto:2.@Retention(RetentionPolicy.SOURCE)):声明注解的有效范围

RetentionPolicy.SOURCE:该注解只在源码中有效

RetentionPolicy.CLASS:该注解源码中和字节码中有效(默认值)

RetentionPolicy.RUNTIME:该注解在源码中,和字节码中有效,运行字节码时也有效